Developing, Deploying, and Debugging Applications on Windows Embedded Standard 7

Contents

Overview

Windows Embedded Standard 7 is the next generation platform within the Windows Embedded Standard portfolio and delivers the power, familiarity, and reliability of the Windows 7 operating system in a highly customizable and componentized form. This article focuses on developers and how they can easily perform their day-to-day tasks in developing applications for Windows Embedded Standard 7.

It may come as a surprise to many that developing an application for *Windows Embedded Standard 7* is not very different from developing the same application for Windows 7. In fact, in most cases, the process is identical. One can write almost any application on Windows 7 and deploy it to *Windows Embedded Standard 7.* Several things related to creating the Windows Embedded Standard 7 image must be observed to make sure that the application works correctly, and has access to all components that it needs in order to run. This article describes how to write such an application from scratch, deploy it to Windows Embedded Standard 7 using the Image Configuration Editor (ICE) tool, Image Builder Wizard (IBW) tool, and how to debug the application should anything go wrong.

This article describes creating a very basic "Hello World" application and deploying it to a Windows Embedded Standard 7 image. The tools that you will need are Visual Studio 2005 or a later version, and the Windows Embedded Standard 7 tools.

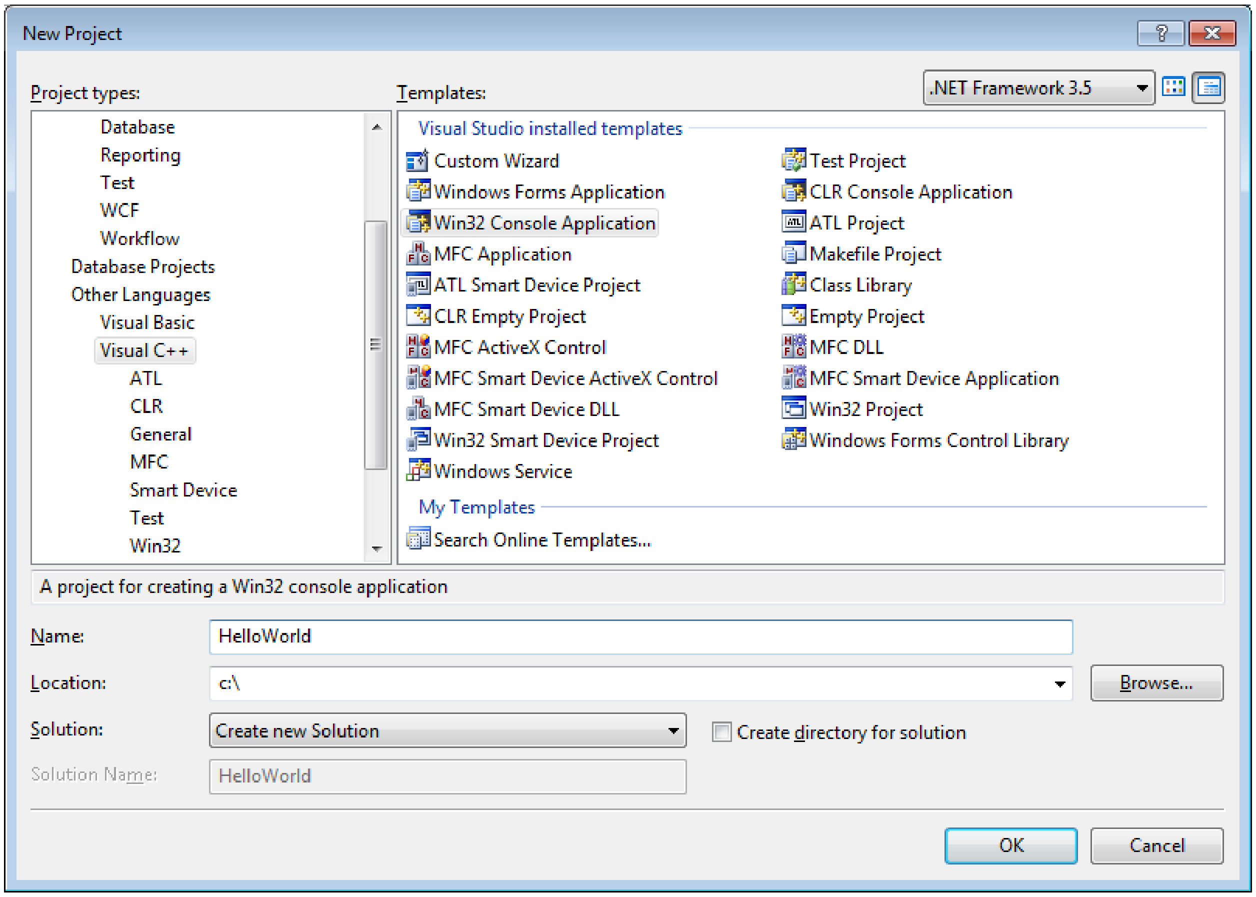
The application

Motivation

This article contains information about how to develop a very simple console-based C++ application that outputs “Hello World” to the console. The article then describes how to deploy and debug the application on a Windows Embedded Standard 7 computer. You can easily apply the knowledge that you gain in deploying and debugging this simple application to a more complex application.

Code and Environment

To create the application, start Visual Studio. On the **File** menu, point to **New**, and then click **Project**. Select a project type by clicking **Visual C++**, and then click **Win32 Console Application**, as shown in Figure 1.


New Project

Project types: Database, Reporting, Test, WCF, Workflow, Database Projects, Other Languages, Visual Basic, Visual C++, ATL, CLR, General, MFC, Smart Device, Test, Win32

Templates: .NET Framework 3.5

Visual Studio installed templates: Custom Wizard, Windows Forms Application, Win32 Console Application, MFC Application, ATL Smart Device Project, CLR Empty Project, MFC ActiveX Control, MFC Smart Device ActiveX Control, MFC Smart Device DLL, Win32 Smart Device Project, Windows Service, Test Project, CLR Console Application, ATL Project, Makefile Project, Class Library, Empty Project, MFC DLL, MFC Smart Device Application, Win32 Project, Windows Forms Control Library

My Templates: Search Online Templates...

A project for creating a Win32 console application

Name: HelloWorld

Location: c:\

Solution: Create new Solution

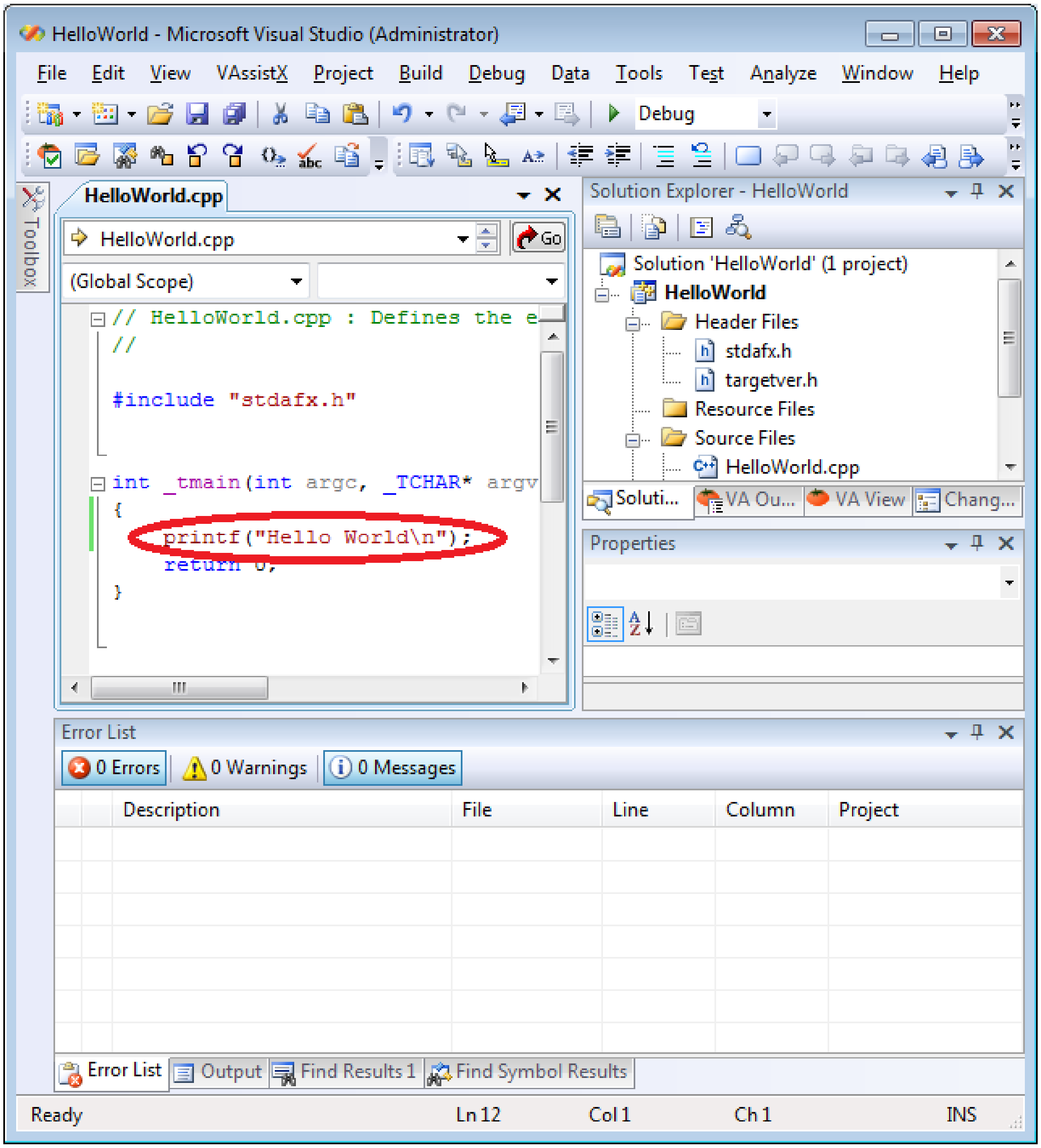
Create directory for solution

Solution Name: HelloWorld

Figure 1

In the **Name** box, type “HelloWorld” then click **OK**. In the **Location** box, type C:\.

In the **Win32 Application Wizard** click Finish. This should create a project similar to the one shown in Figure 2.



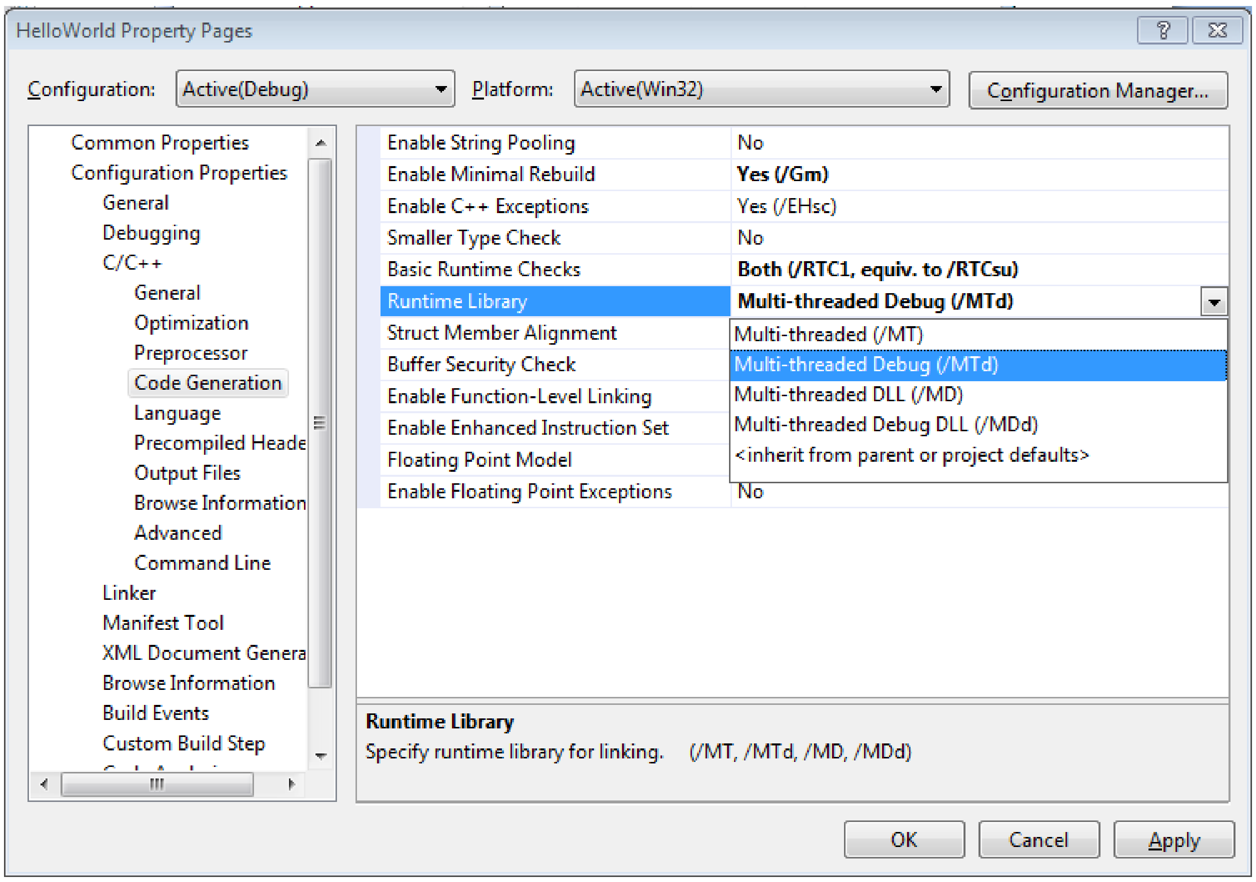
```
// HelloWorld.cpp : Defines the e
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv
{
    printf("Hello World\n");
    return 0;
}
```

Figure 2

As shown in Figure 1.2, type the "**printf("Hello World\n")**" code. To avoid having dynamic dependencies you must change one aspect of the application by right-clicking the HelloWorld project in Solution Explorer then clicking **Properties**. Expand **Configuration Properties**, and then expand **C/C++**. Click **Code Generation**. Change Runtime Library from Multi-threaded Debug DLL (/MDd) to Multi-threaded Debug (/MTd) as shown in Figure 3.



Enable String Pooling	No
Enable Minimal Rebuild	**Yes (/Gm)**
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	**Both (/RTC1, equiv. to /RTCsu)**
Runtime Library	**Multi-threaded Debug (/MTd)**
Struct Member Alignment	
Buffer Security Check	
Enable Function-Level Linking	
Enable Enhanced Instruction Set	
Floating Point Model	
Enable Floating Point Exceptions	No

Runtime Library
Specify runtime library for linking. (/MT, /MTd, /MD, /MDd)

Figure 3

Changing the runtime library to **Multi-threaded Debug** makes the application self-contained without the need for additional DLLs. The application is finished. Press **CTRL+F5** to test the application. You should see a console window that displays "Hello World".

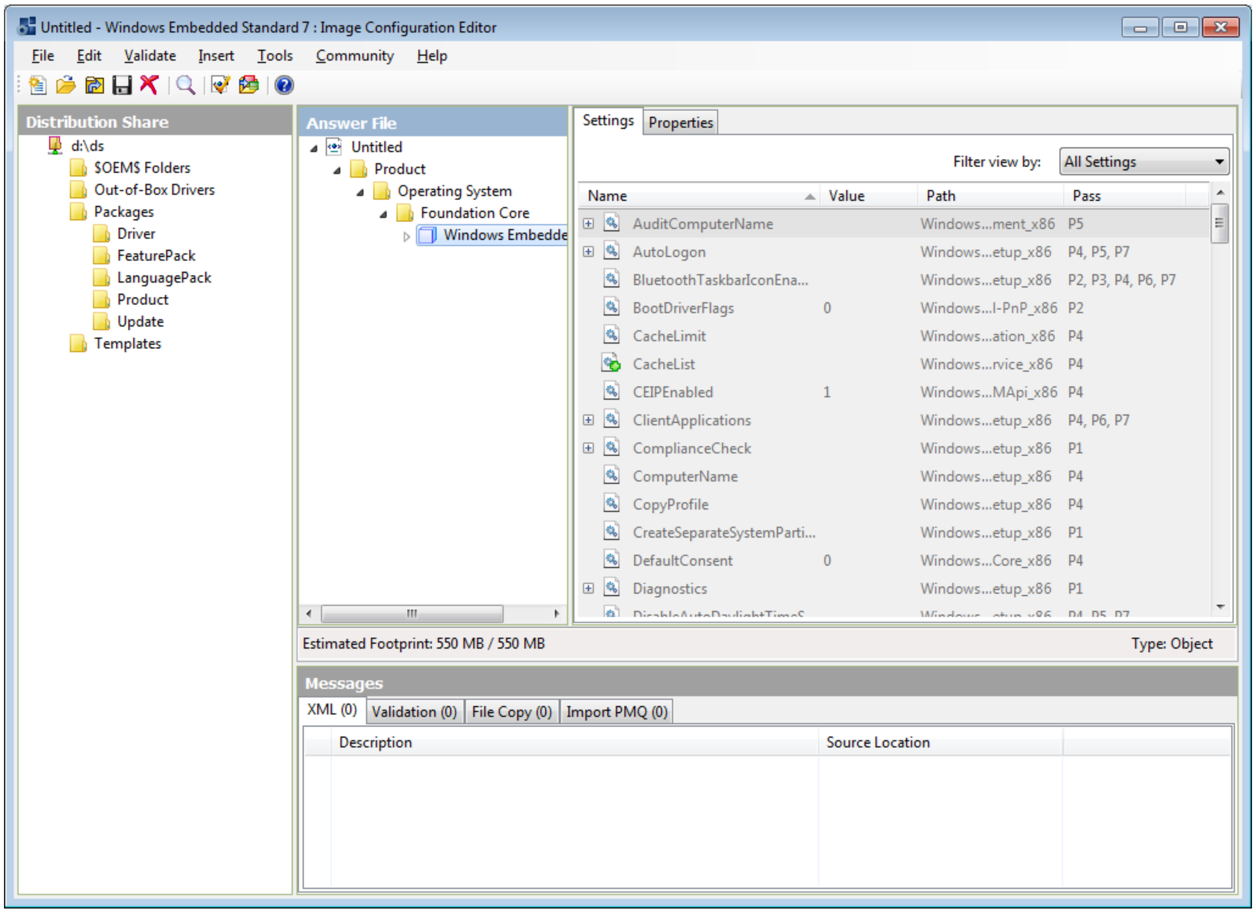
Preparing the Windows Embedded Standard 7 Image

Now that the application is finished, your goal will be to create an image with the minimum Windows Embedded Standard footprint required to deploy this application. To create the image, follow these steps.

Statically analyze the HelloWorld application

To start the Image Configuration Editor (ICE), click **Start**, point to **All Programs**, point to **Windows Embedded Standard 7**, and then click **Image Configuration Editor**. Then open the 32-bit Distribution

Share by pointing to File and clicking **Select Distribution Share**. On the **File** menu, click **New Answer File**. You should now have a screen similar to the one shown in Figure 4.



Name	Value	Path	Pass
AuditComputerName		Windows...ment_x86	P5
AutoLogon		Windows...etup_x86	P4, P5, P7
BluetoothTaskbarIconEna...		Windows...etup_x86	P2, P3, P4, P6, P7
BootDriverFlags	0	Windows...l-PnP_x86	P2
CacheLimit		Windows...ation_x86	P4
CacheList		Windows...rvice_x86	P4
CEIPEnabled	1	Windows...MApi_x86	P4
ClientApplications		Windows...etup_x86	P4, P6, P7
ComplianceCheck		Windows...etup_x86	P1
ComputerName		Windows...etup_x86	P4
CopyProfile		Windows...etup_x86	P4
CreateSeparateSystemParti...		Windows...etup_x86	P1
DefaultConsent	0	Windows...Core_x86	P4
Diagnostics		Windows...etup_x86	P1

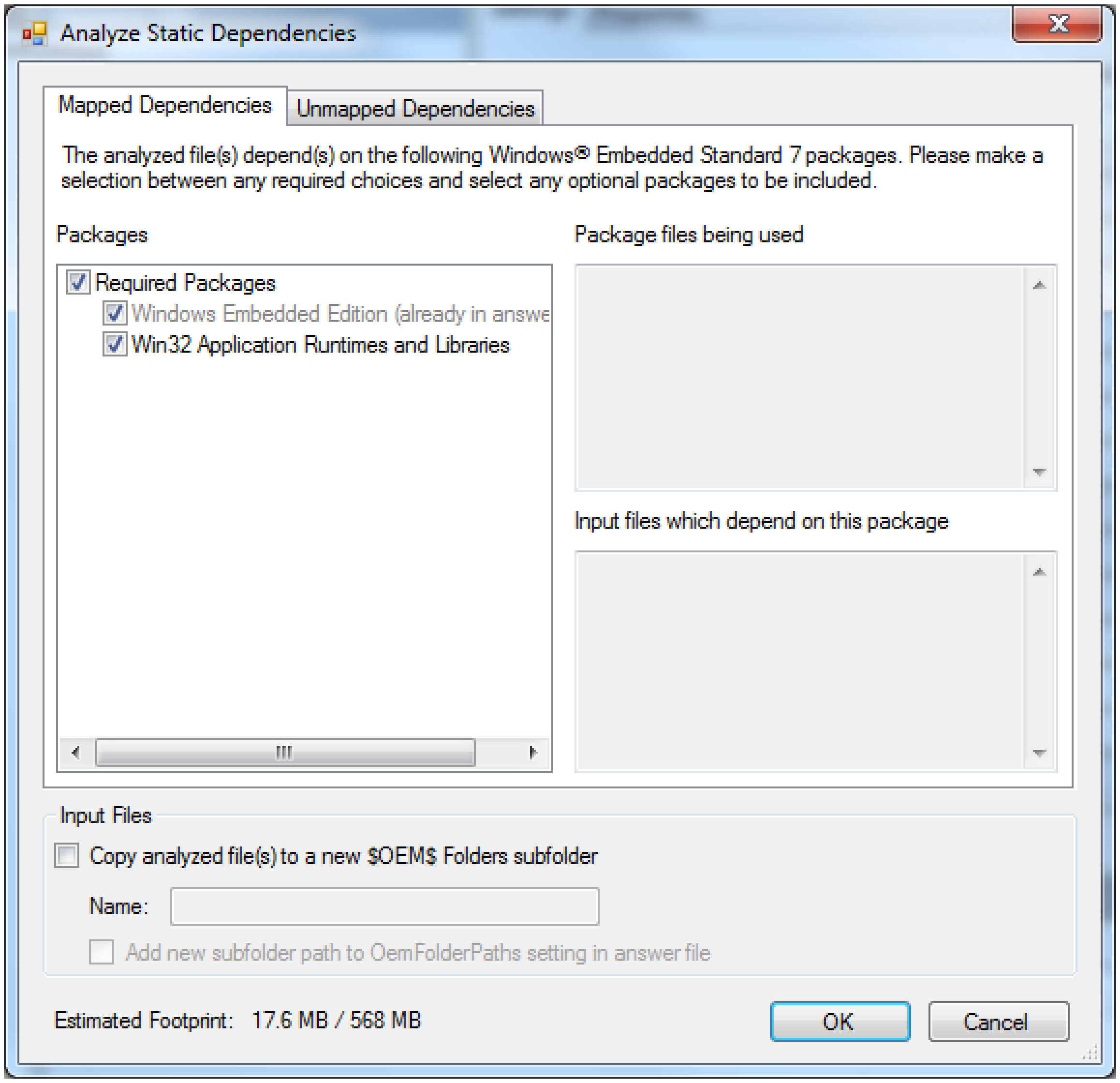
Estimated Footprint: 550 MB / 550 MB

Type: Object

Figure 4

Now that you have an empty answer file, you will do a static analysis of the HelloWorld.exe binary. On the Tools menu, click **Analyze Static Dependencie**s. The Windows File Open dialog box appears. Click the **Files of type arrow**, and select **Executables and Libraries (*.dll, *.exe)**.

Locate the HelloWorld.exe file that is located in C:\HelloWorld\Debug folder, and then click **Open**. The Analyze Static Dependencies dialog box appears as shown in Figure 5.



The analyzed file(s) depend(s) on the following Windows® Embedded Standard 7 packages. Please make a selection between any required choices and select any optional packages to be included.

Packages

- [✓] Required Packages
  - [✓] Windows Embedded Edition (already in answe
  - [✓] Win32 Application Runtimes and Libraries

Package files being used

Input files which depend on this package

Input Files

- [ ] Copy analyzed file(s) to a new OEM Folders subfolder

Name:

- [ ] Add new subfolder path to OemFolderPaths setting in answer file

Estimated Footprint: 17.6 MB / 568 MB

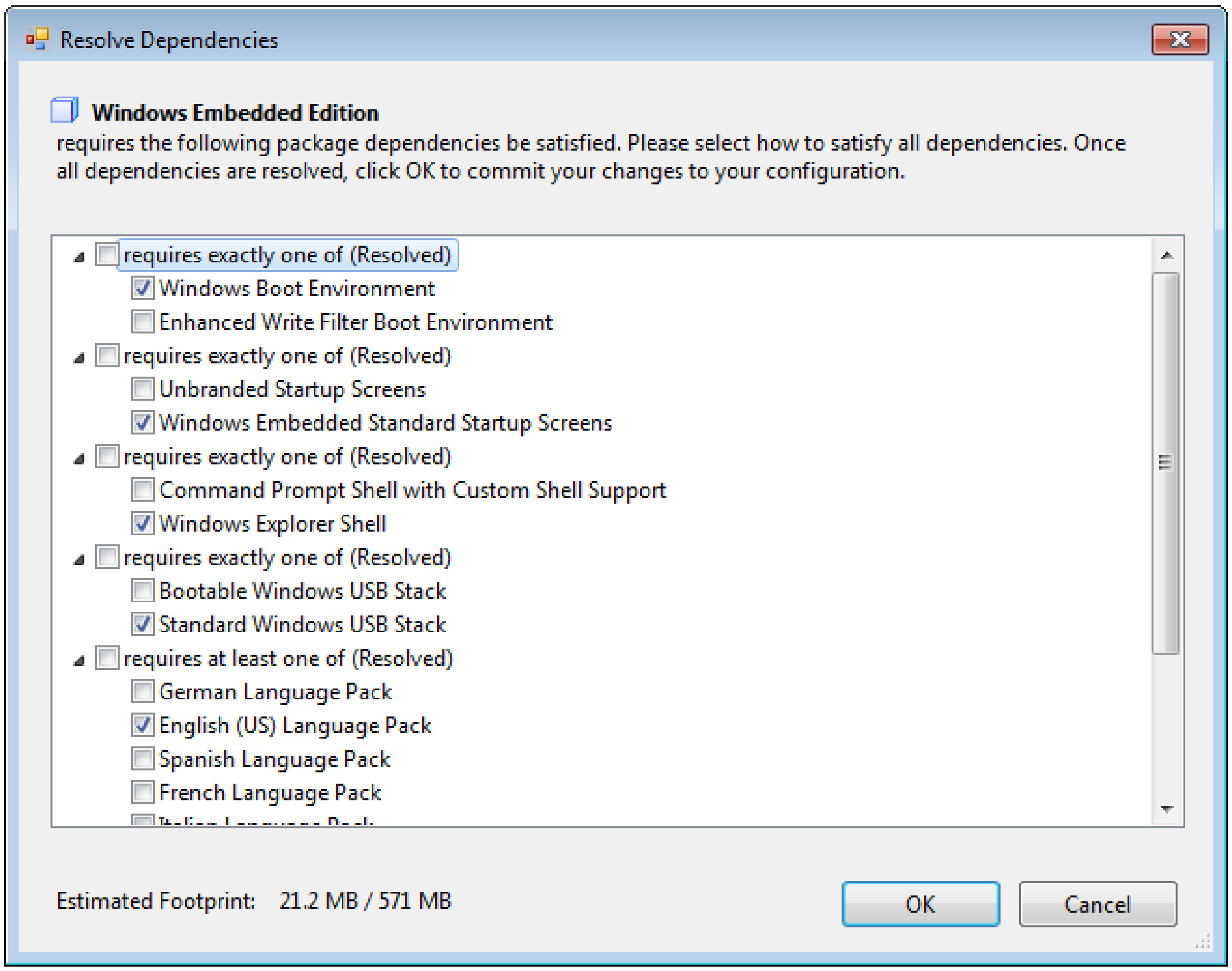
Figure 5

Be aware that the required packages are already in the answer file.

Usually when you create an image and want the application to run on the image, you would select the **Copy Analyzed File(s) to a new $OEM Folders subfolder** check box. However, because the goal of the current image is to actually debug the application, you will skip this step. If you were actually going to deploy the application, you would select this option. After vetting and debugging the application, you can return to this step and select that option to correctly deploy the binary to the target image. Clear the **Copy Analyzed File(s) to a new $OEM Folders subfolder** check box, and then click **OK**.

1. Resolve dependencies

Resolve all image dependencies by pressing **CTRL+F5** in ICE. One error will be displayed in the Messages pane. Double-click the error and the **Resolve Dependencies** dialog box appears as shown in Figure 6.



Resolve Dependencies

Windows Embedded Edition

requires the following package dependencies be satisfied. Please select how to satisfy all dependencies. Once all dependencies are resolved, click OK to commit your changes to your configuration.

- [ ] requires exactly one of (Resolved)
  - [✓] Windows Boot Environment
  - [ ] Enhanced Write Filter Boot Environment
- [ ] requires exactly one of (Resolved)
  - [ ] Unbranded Startup Screens
  - [✓] Windows Embedded Standard Startup Screens
- [ ] requires exactly one of (Resolved)
  - [ ] Command Prompt Shell with Custom Shell Support
  - [✓] Windows Explorer Shell
- [ ] requires exactly one of (Resolved)
  - [ ] Bootable Windows USB Stack
  - [✓] Standard Windows USB Stack
- [ ] requires at least one of (Resolved)
  - [ ] German Language Pack
  - [✓] English (US) Language Pack
  - [ ] Spanish Language Pack
  - [ ] French Language Pack

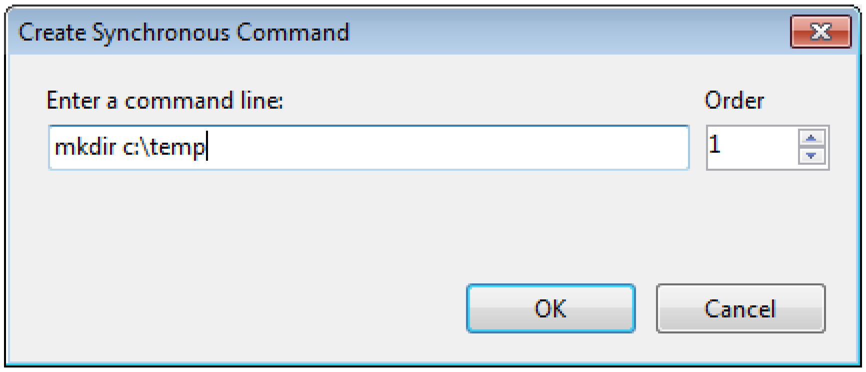
Estimated Footprint: 21.2 MB / 571 MB

Figure 6

Select the options shown in Figure 6.

2. Create shared folder for the target device.

In order to be able to run the application on the target device, you will create a shared folder. On the **Insert** menu, point to **Synchronous Command**, and then click **Pass 7 oobeSystem** . The Create Synchronous Command dialog box will appear as shown in Figure 7:



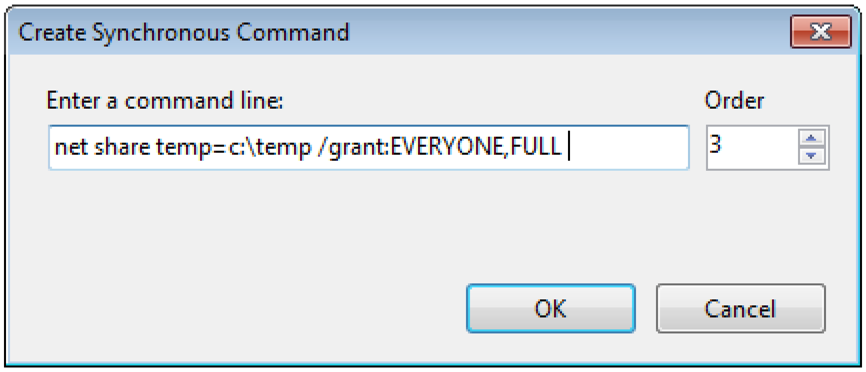
```
mkdir c:\temp
```

Order: 1

Figure 7

In **Enter a command line**, type **mkdir c:\temp.** Click OK. When the command runs it will create the folder where your application will be deployed.

Add another synchronous command to be able to dynamically deploy the program to the target computer. On the **Insert** menu, point to **Synchronous Command**, and then click **Pass 7 oobeSystem** . Type the following command: net share temp=c:\temp /grant:EVERYONE,FULL as shown in Figure 8:



```
net share temp=c:\temp /grant:EVERYONE,FULL
```

Order: 3

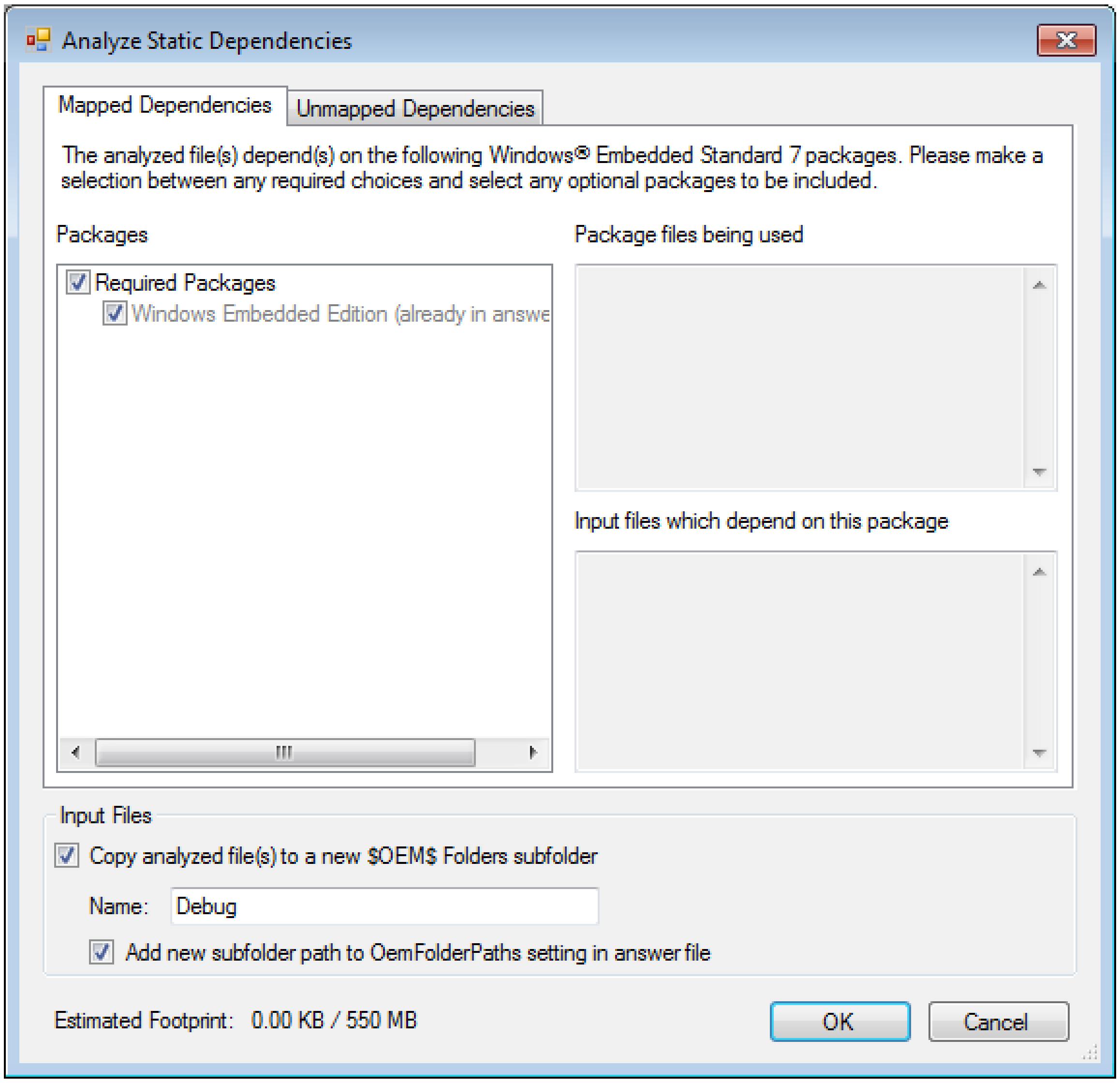
Figure 8

3. Add Visual Studio Debugger to image.

Because the goal of this exercise is not only to have the application up and running, but also to set up remote debugging should anything go wrong, you must install the Visual Studio debugging client onto our image. The remote debugging client is located on the installation media of Visual Studio 2005 or a later version. On Visual Studio 2008 installation media, it is located at Remote

Debugger\x86\rdbgsetup.exe. Perform a static analysis on this item and add it to our image. On the **Tools** menu, click **Analyze Static Dependencies** in ICE.

The static analysis will reveal that the item will need nothing more than the Windows Embedded Core package. Check the **Copy analyzed File(s) into OEM Folder subfolder** check box. In Name, type Debug, and then select Add new subfolder path to OemFolderPaths as shown in Figure 9:



The analyzed file(s) depend(s) on the following Windows® Embedded Standard 7 packages. Please make a selection between any required choices and select any optional packages to be included.

- [✓] Required Packages
  - [✓] Windows Embedded Edition (already in answe

Input Files

[✓] Copy analyzed file(s) to a new OEM Folders subfolder

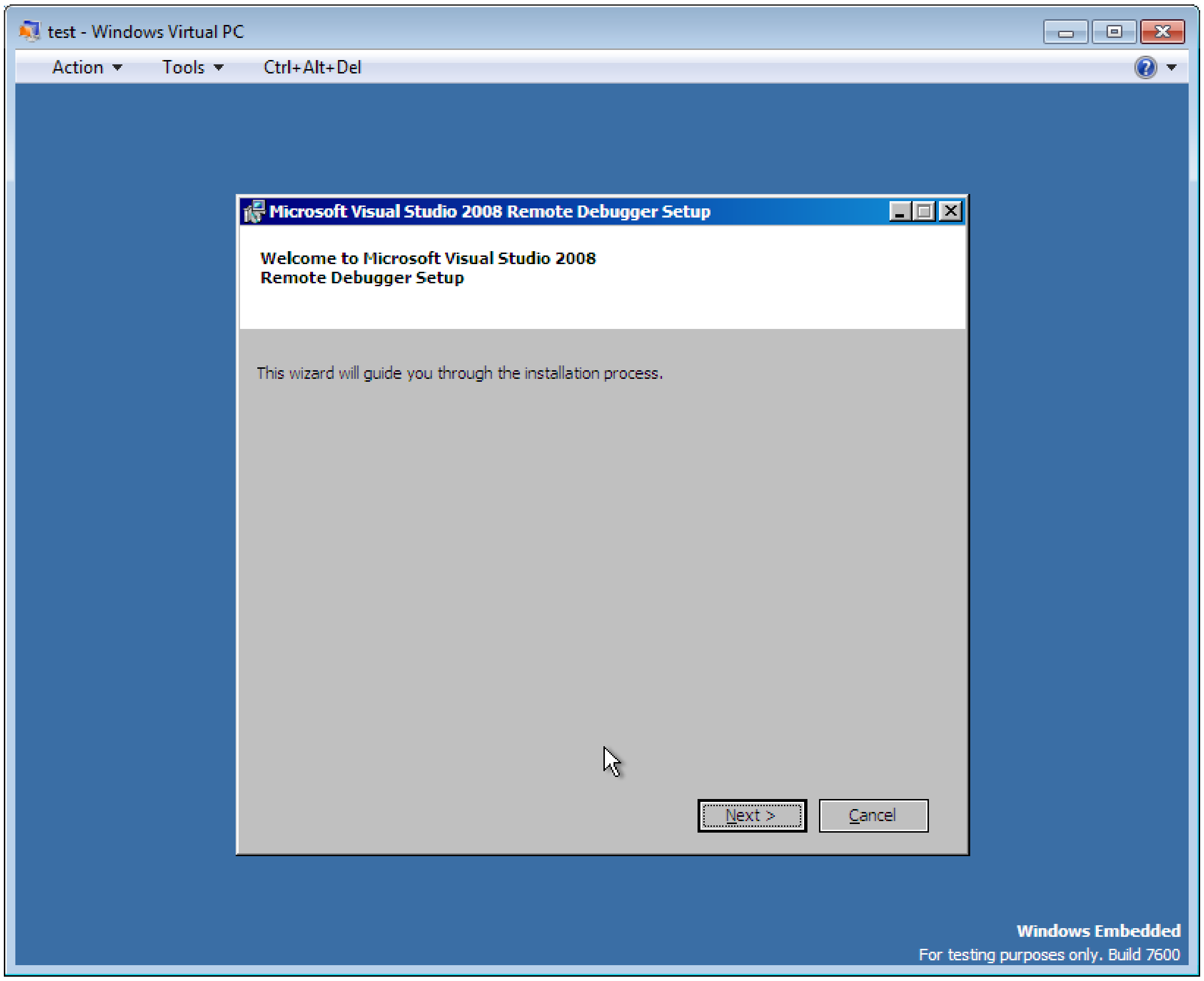
Name: Debug

[✓] Add new subfolder path to OemFolderPaths setting in answer file

Estimated Footprint: 0.00 KB / 550 MB

Figure 9

There is one additional step that you will take to install the remote debugger with product. On the **Insert** menu, point to **Synchronous Command**, and then click **Pass 7 oobeSystem** . In **Enter a command line**, type "**C:\rdbgsetup.exe**" for the remote debugger setup. This command will run the remote debugger setup during the first boot run of the image as shown in Figure 10.


test - Windows Virtual PC
Action
Tools
Ctrl+Alt+Del
Microsoft Visual Studio 2008 Remote Debugger Setup
Next >
Cancel
Windows Embedded
For testing purposes only. Build 7600

Welcome to Microsoft Visual Studio 2008
Remote Debugger Setup

This wizard will guide you through the installation process.

Figure 10

4. Create IBW Image

Now you have prepared an answer file, you can create an IBW image with only the required packages. Insert a USB key into your computer. In ICE, on the **Tools** menu, point to **Create Media**, and then click **Create IBW Image from Answerfile**. The Create IBW Disk dialog box appears. Locate the root folder of the USB key. Click **OK**. The files that are needed to create the image will be copied to our USB drive as shown in Figure 11.

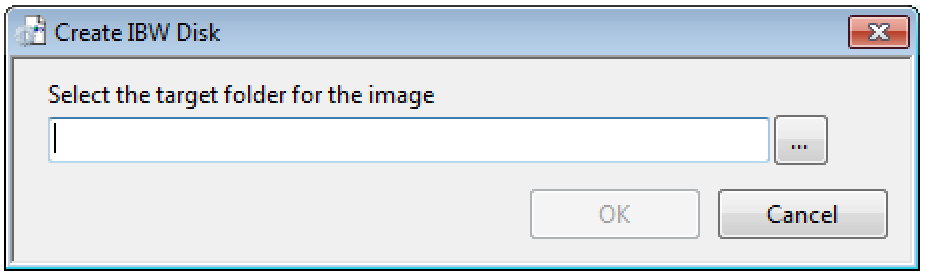
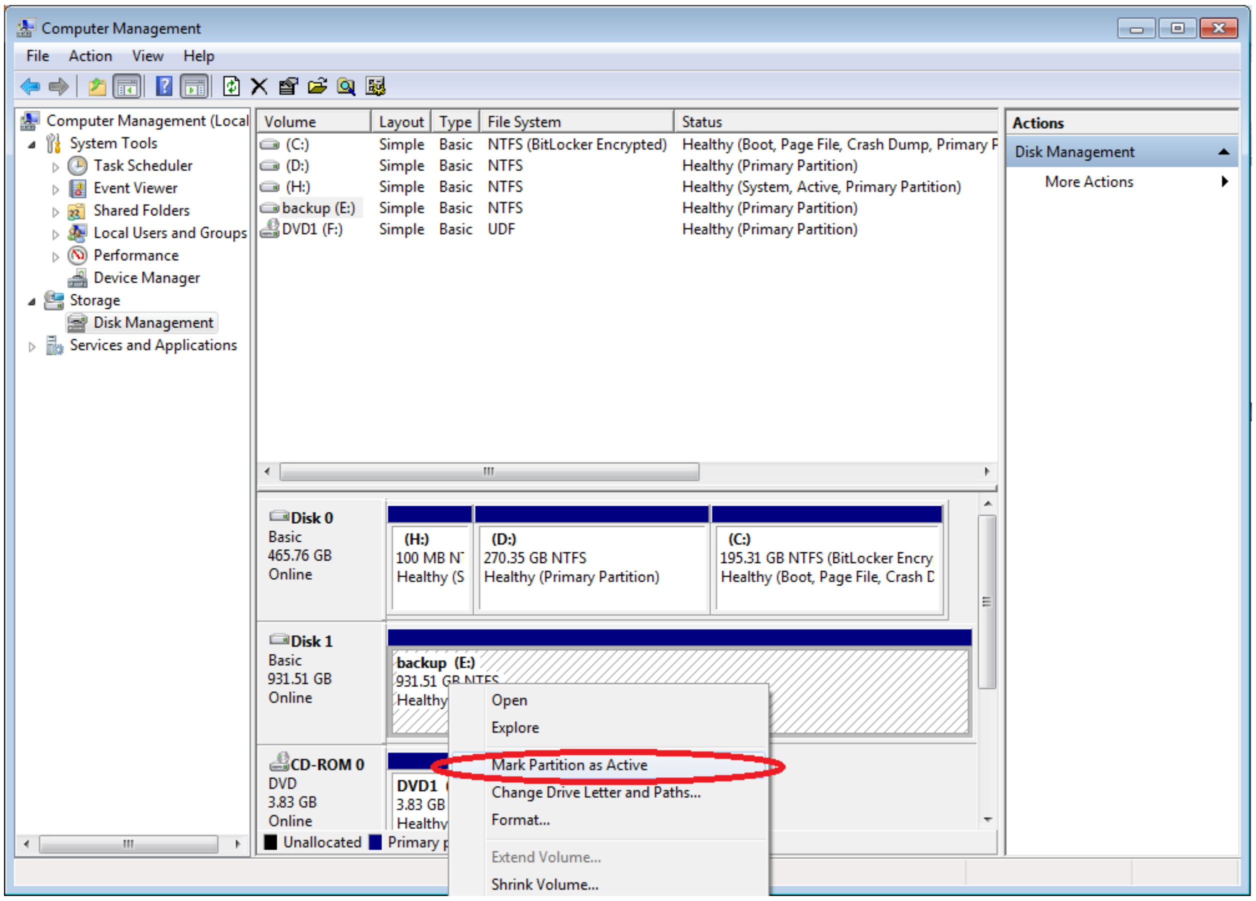


Figure 11

Deploying

Before you create the image by using IBW, make sure that the primary partition of the USB keys set to active so that the image can boot from the key. You will use the Disk Management utility as shown in Figure 12. Click **Start**, and type “Computer Management”. Press ENTER. The Computer Management utility will appear as shown in figure 12. Right-click the USB partition and then click **Mark Partition as Active**.



Volume	Layout	Type	File System	Status
(C:)	Simple	Basic	NTFS (BitLocker Encrypted)	Healthy (Boot, Page File, Crash Dump, Primary P
(D:)	Simple	Basic	NTFS	Healthy (Primary Partition)
(H:)	Simple	Basic	NTFS	Healthy (System, Active, Primary Partition)
backup (E:)	Simple	Basic	NTFS	Healthy (Primary Partition)
DVD1 (F:)	Simple	Basic	UDF	Healthy (Primary Partition)

Disk 0
Basic
465.76 GB
Online

(H:)
100 MB N
Healthy (S

(D:)
270.35 GB NTFS
Healthy (Primary Partition)

(C:)
195.31 GB NTFS (BitLocker Encry
Healthy (Boot, Page File, Crash D

Disk 1
Basic
931.51 GB
Online

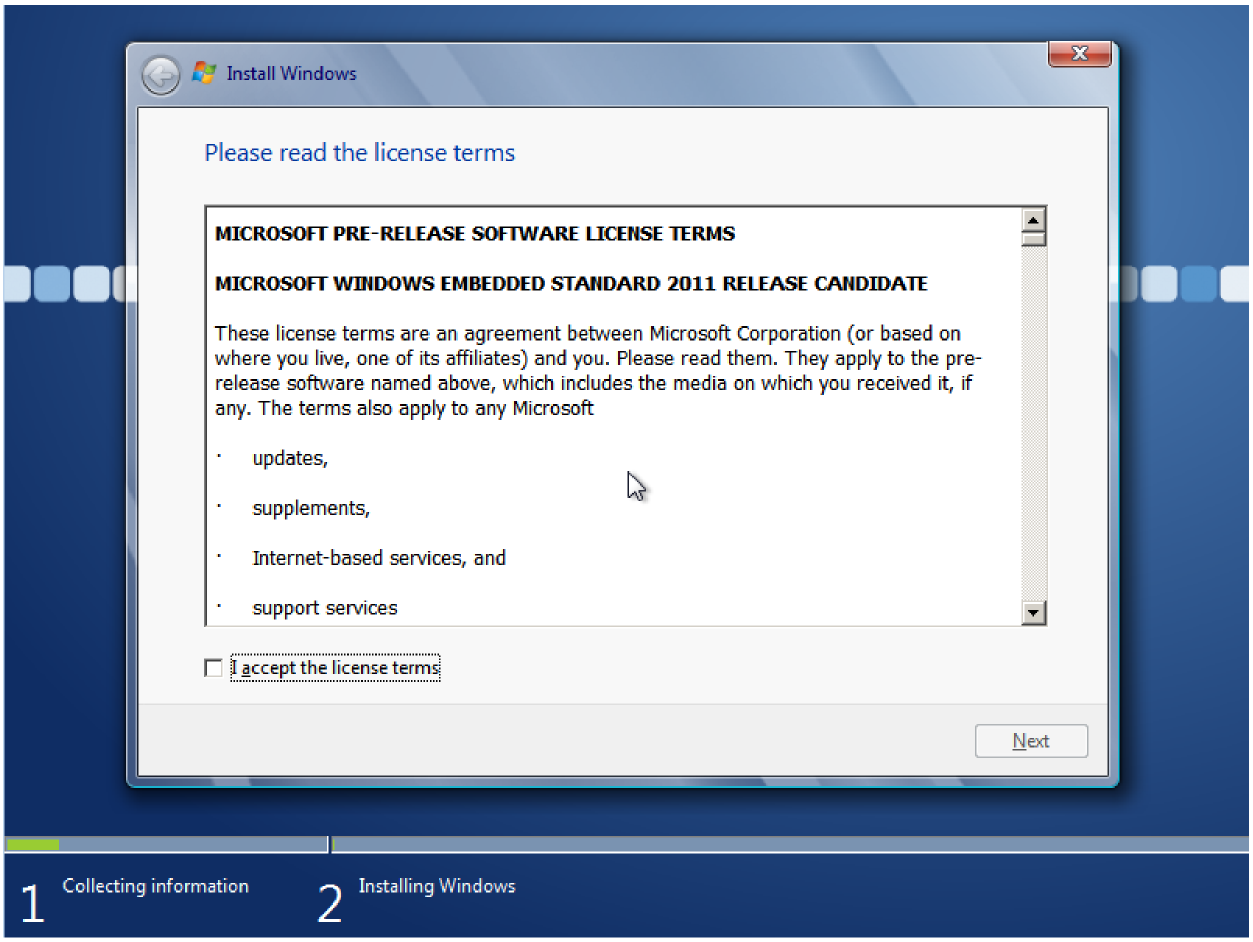
backup (E:)
931.51 GB NTFS
Healthy

CD-ROM 0
DVD
3.83 GB
Online

DVD1
3.83 GB
Healthy

Figure 12

Make sure that the BIOS of the device is set so that the USB device is given boot priority. Plug the USB key into the device and restart it. IBW appears as shown in Figure 13.



MICROSOFT PRE-RELEASE SOFTWARE LICENSE TERMS

MICROSOFT WINDOWS EMBEDDED STANDARD 2011 RELEASE CANDIDATE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the pre-release software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft

· updates,

· supplements,

· Internet-based services, and

· support services

Figure 13

Accept the license, and accept all the default options, or other options depending on what your needs are. These selections will not affect the ability to complete the next steps in this document. After the image is deployed, you should have a display similar to the one in Figure 14.

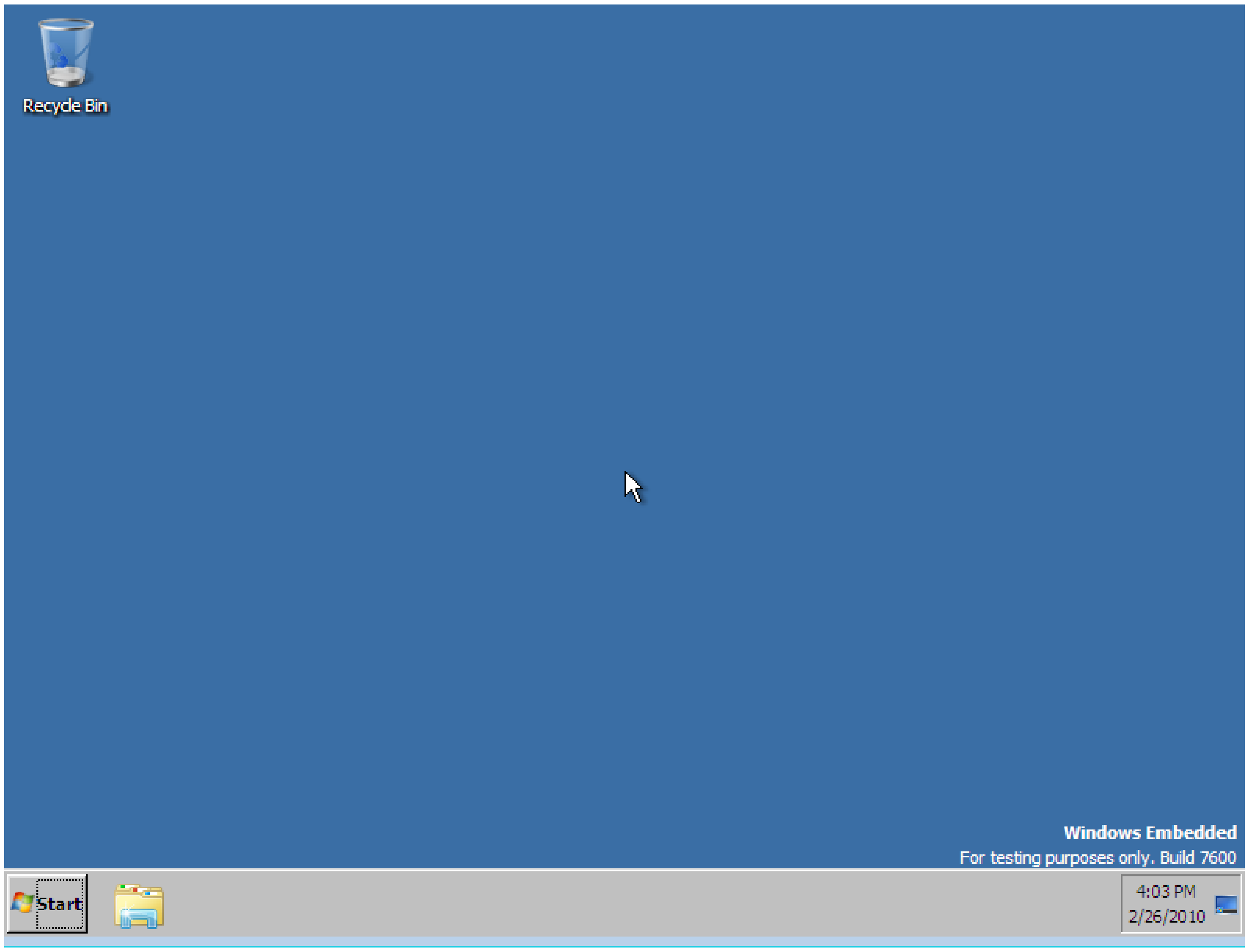
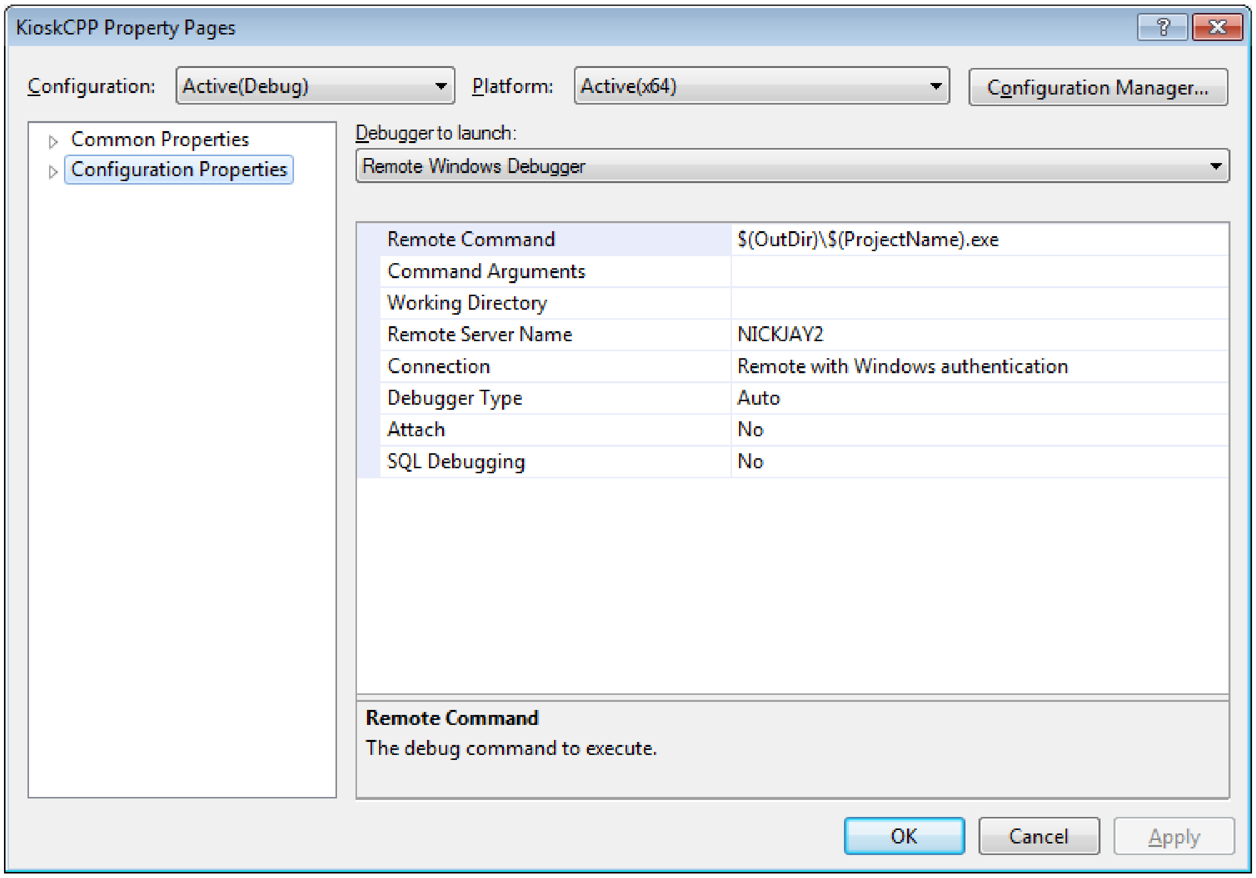


Figure 14

Debugging

Now that the image deployed, it is fairly easy to remotely-deploy our application. First in Visual Studio, instead of outputting the application locally, you will output it to the embedded computer that we created. In Visual Studio, on the **Project** menu, click **Properties**. The **Project Properties** page appears as shown in Figure 15:



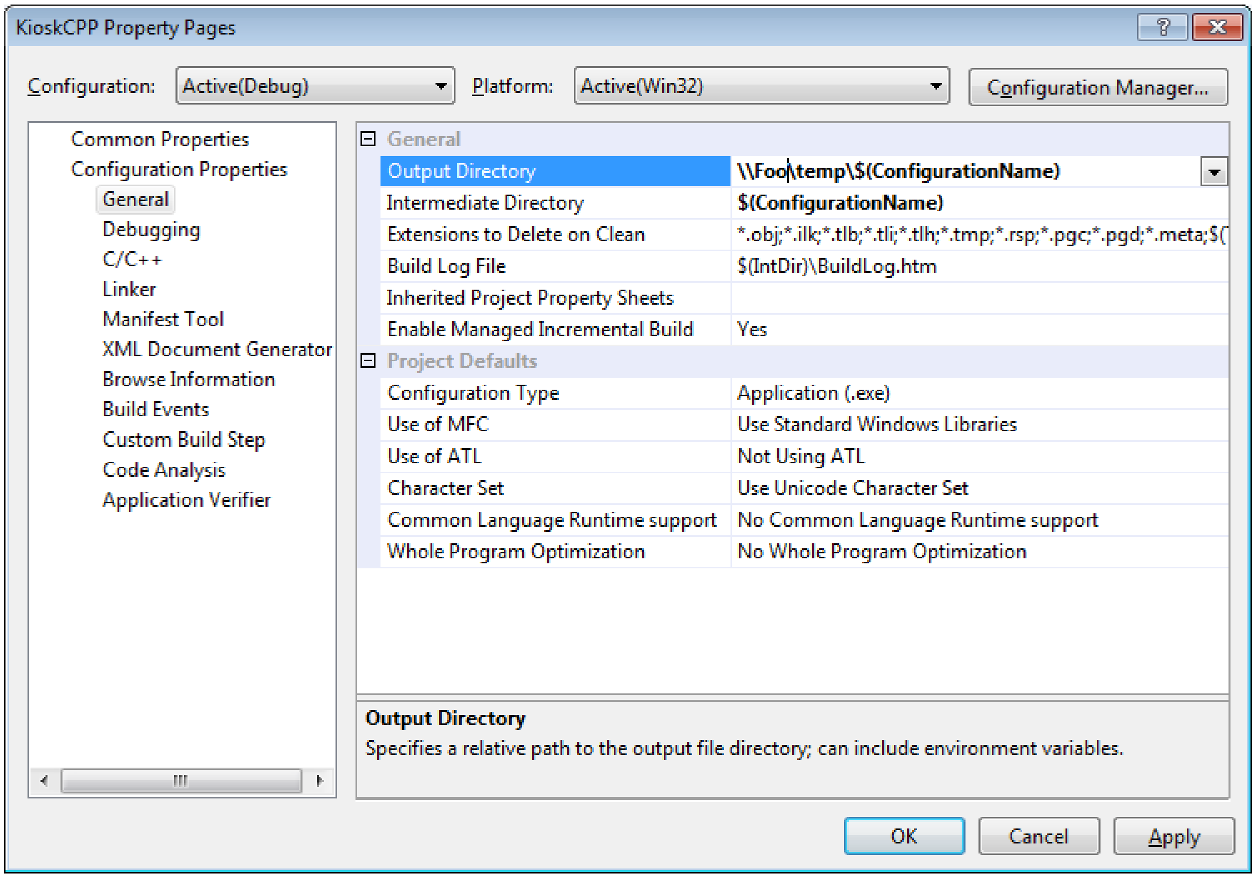
Debugger to launch: Remote Windows Debugger

Remote Command	$(OutDir)\$(ProjectName).exe
Command Arguments	
Working Directory	
Remote Server Name	NICKJAY2
Connection	Remote with Windows authentication
Debugger Type	Auto
Attach	No
SQL Debugging	No

Remote Command
The debug command to execute.

Figure 15

Expand **Configuration Properties,** and then click **General**. Change the default **Output Directory** to the target computer's temp folder. For example, if the name of computer is **Foo**, then you would type **\\Foo\temp\$(ConfigurationName)**. This is shown in Figure 16:

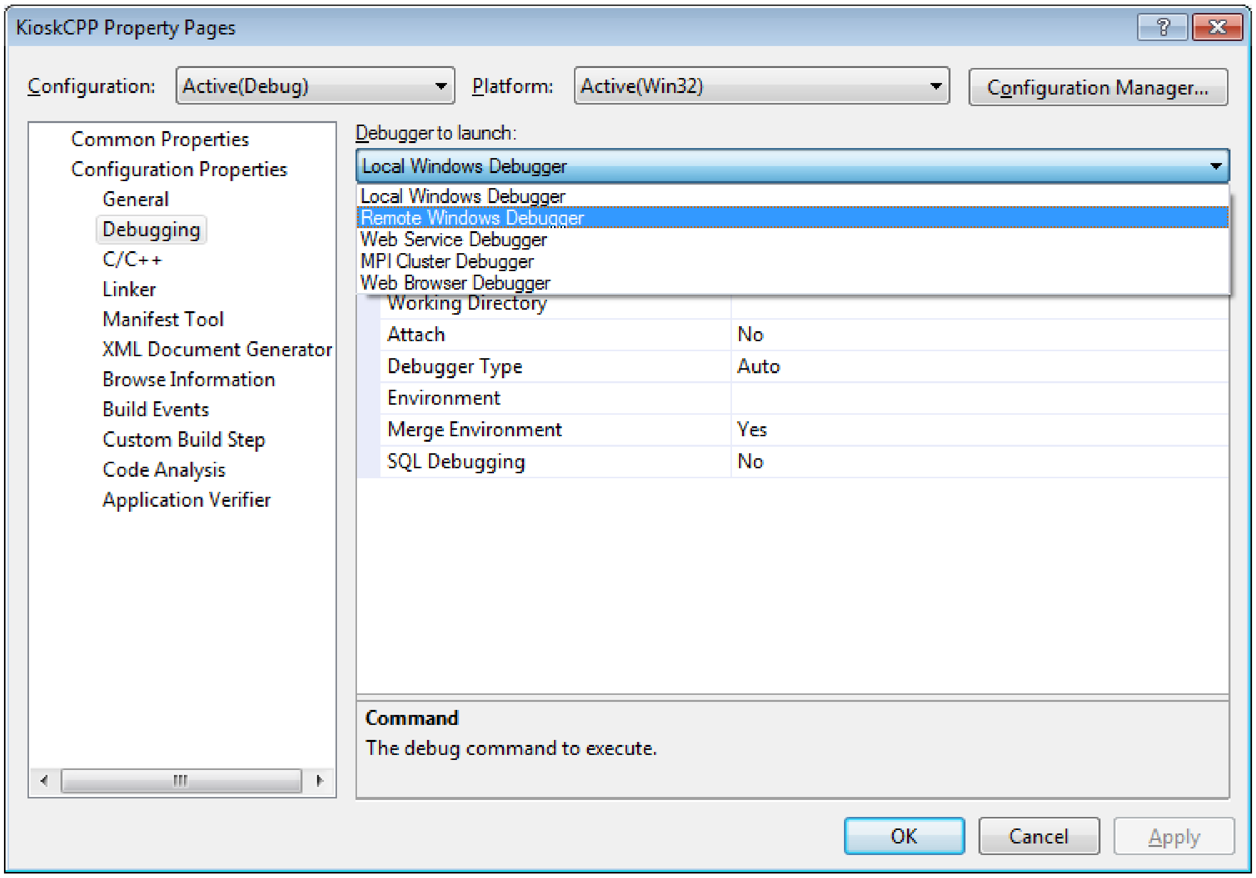


General	
Output Directory	\\Foo\temp\$(ConfigurationName)
Intermediate Directory	$(ConfigurationName)
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;$(
Build Log File	$(IntDir)\BuildLog.htm
Inherited Project Property Sheets	
Enable Managed Incremental Build	Yes
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Character Set	Use Unicode Character Set
Common Language Runtime support	No Common Language Runtime support
Whole Program Optimization	No Whole Program Optimization

Output Directory
Specifies a relative path to the output file directory; can include environment variables.

Figure 16

From the **Configuration Properties** list, click **Debugging**. The default option in Visual Studio is to use the local Debugger. Select the option to use the remote debugger by selecting the **Remote Windows Debugger** check box as shown in Figure 17:

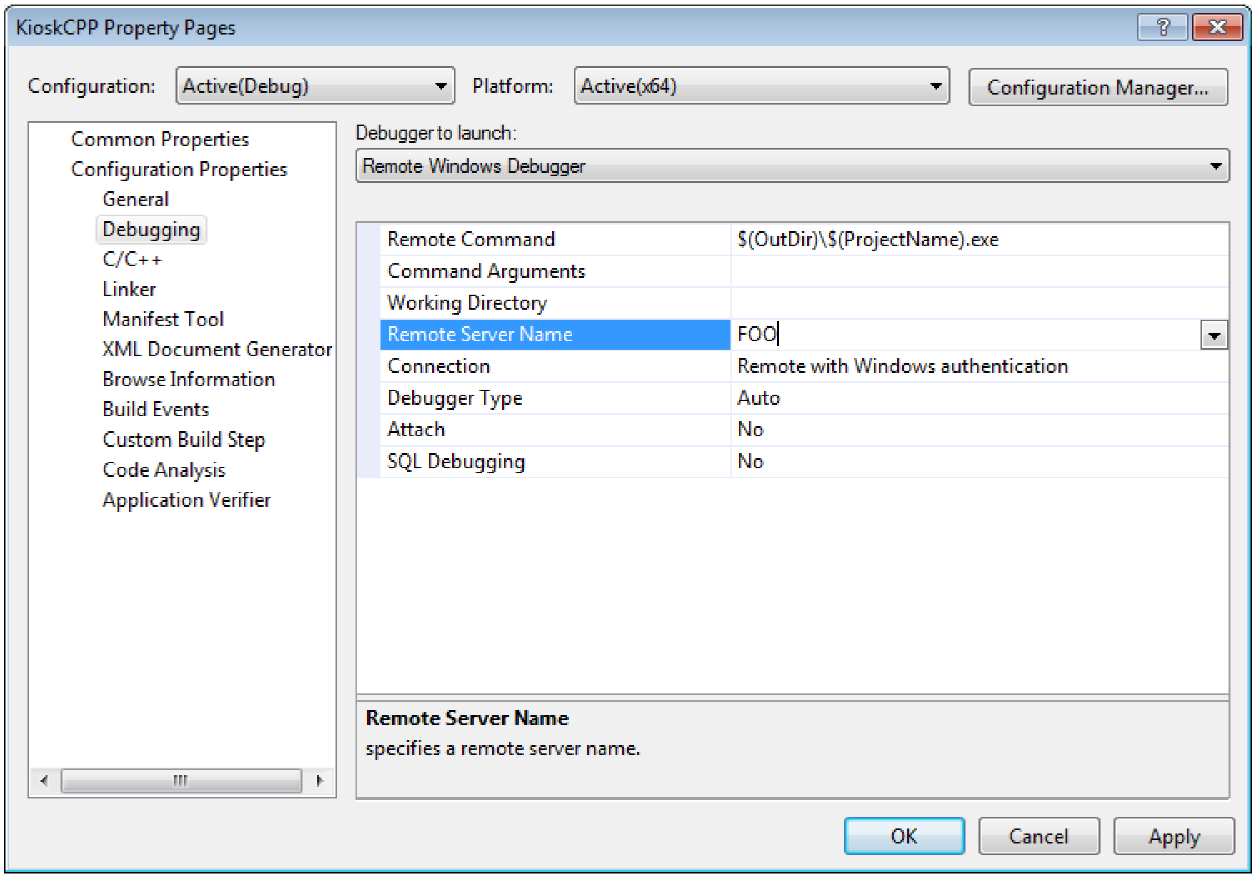


Working Directory	
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No

Command
The debug command to execute.

Figure 17

For the Remote command, type **$(OutDir)\$(ProjectName).exe**. This will start our program on the remote computer. Change the **Remote Server Name** and **Connection** options to match the computer name and authentication that is used for your target image. It will vary depending on whether you are connected to a domain or part of a workgroup. This is shown in Figure 18.


KioskCPP Property Pages

Configuration: Active(Debug) Platform: Active(x64) Configuration Manager...

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis
  - Application Verifier

Debugger to launch: Remote Windows Debugger

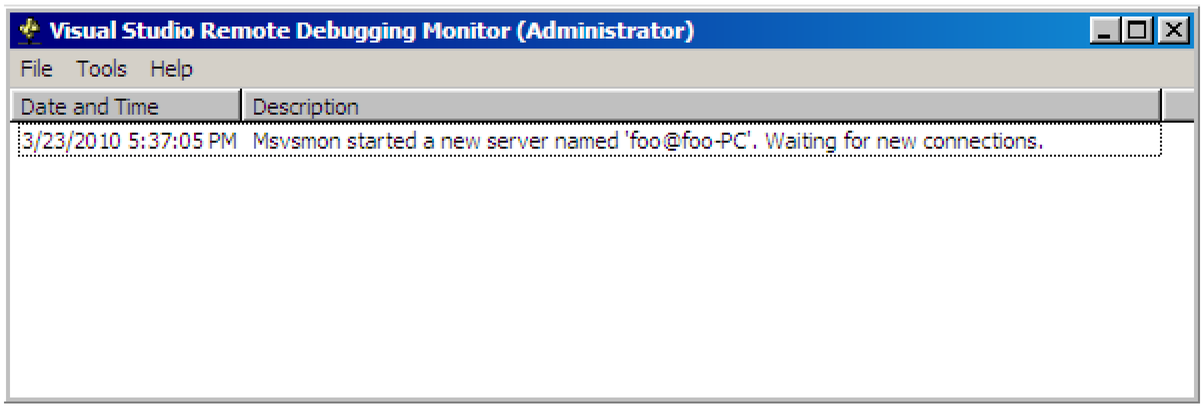
Remote Command	$(OutDir)\$(ProjectName).exe
Command Arguments	
Working Directory	
Remote Server Name	FOO
Connection	Remote with Windows authentication
Debugger Type	Auto
Attach	No
SQL Debugging	No

Remote Server Name
specifies a remote server name.

OK Cancel Apply

Figure 18

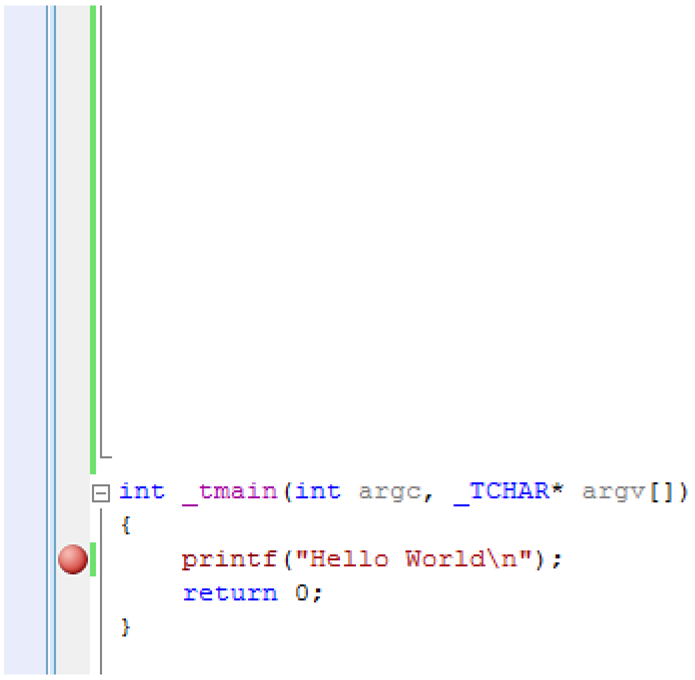
You are almost finished. On the target computer, run the Remote Debugging Client. On the target computer click **Start**, point to **Programs**, point to **Microsoft Visual Studio 2008**. Point to **Visual Studio Tools**, and then click **Visual Studio 2008 Remote Debugger**. The **Visual Studio Remote Debugging Monitor** appears as shown in Figure 19:



```
3/23/2010 5:37:05 PM  Msvsmon started a new server named 'foo@foo-PC'. Waiting for new connections.
```

Figure 19

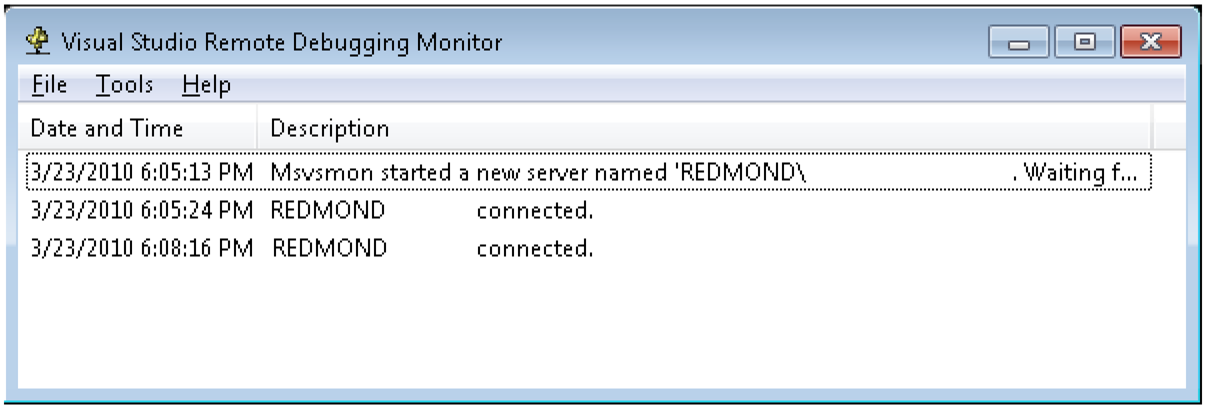
You are now ready to deploy the application directly from the development computer to the destination computer and debug it. In the C++ project, set a breakpoint where the “Hello World” is displayed on the console, as shown in Figure 20:



```
int _tmain(int argc, _TCHAR* argv[])
{
    printf("Hello World\n");
    return 0;
}
```

Figure 20

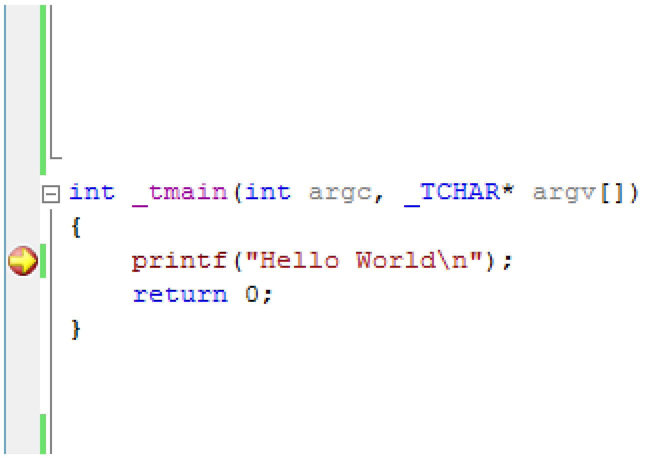
On the **Debug** menu, click **Start Debugging.** This starts the application on the destination computer. You will receive a notification in the remote debugging client on the destination Windows Embedded Standard 7 computer that an external computer has connected as shown in Figure 21:



Date and Time	Description
3/23/2010 6:05:13 PM	Msvsmon started a new server named 'REDMOND\ . Waiting f...
3/23/2010 6:05:24 PM	REDMOND connected.
3/23/2010 6:08:16 PM	REDMOND connected.

Figure 21

When you run the application, it will hit the breakpoint you set on the development computer as shown in Figure 22:



```
int _tmain(int argc, _TCHAR* argv[])
{
    printf("Hello World\n");
    return 0;
}
```

Figure 22

Summary

The techniques described in this document apply not only to Visual C++ applications, but to any other application type you can create in Windows 7. This includes, but is not limited to, the following: ASP.NET pages, Windows Forms Applications, Windows Presentation Framework (WPF) applications, Silverlight applications, Windows Communication Framework (WCF) services, and more. In addition, although it is beyond the scope of this document to describe in detail, third-party and open-source technologies that can run on Windows 7 can also run on Windows Embedded Standard 7. Because the correct dependencies are satisfied on the target image, there really is no limit to the applications that you can develop, deploy, and debug on Windows Embedded Standard 7.